LPI exam prep: System security

Intermediate Level Administration (LPIC-2) topic 212

Skill Level: Intermediate

David Mertz (mertz@gnosis.cx)
Developer
Gnosis Software, Inc.

13 Jun 2006

In this tutorial, the sixth in a series of seven tutorials covering intermediate network administration on Linux®, David Mertz continues preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 202. By necessity, this tutorial touches briefly on a wide array of Linux-related topics from a security-conscious network server perspective, including general issues of routing, firewalls, and NAT translation and the relevant tools. It addresses setting security policies for FTP and SSH; reviews general access control with tcpd, hosts.allow, and friends; and presents some basic security monitoring tools and shows where to find security resources.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at two levels: *junior level* (also called "certification level 1") and *intermediate level* (also called "certification level 2"). To attain certification level 1, you must pass exams 101 and 102; to attain certification level 2, you must pass exams 201 and 202.

developerWorks offers tutorials to help you prepare for each of the four exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. For LPI exam 202, the seven topics and corresponding developerWorks tutorials are:

Table 1. LPI exam 202: Tutorials and topics

LPI exam 202 topic	developerWorks tutorial	Tutorial summary
Topic 205	LPI exam 202 prep (topic 205): Networking configuration	Learn how to configure a basic TCP/IP network, from the hardware layer (usually Ethernet, modem, ISDN, or 802.11) through the routing of network addresses.
Topic 206	LPI exam 202 prep (topic 206): Mail and news	Learn how to use Linux as a mail server and as a news server. Learn about mail transport, local mail filtering, mailing list maintenance software, and server software for the NNTP protocol.
Topic 207	LPI exam 202 prep (topic 207): DNS	Learn how to use Linux as a DNS server, chiefly using BIND. Learn how to perform a basic BIND configuration, manage DNS zones, and secure a DNS server.
Topic 208	LPI exam 202 prep (topic 208): Web services	Learn how to install and configure the Apache Web server, and learn how to implement the Squid proxy server.
Topic 210	LPI exam 202 prep (topic 210): Network client management	Learn how to configure a DHCP server, an NIS client and server, an LDAP server, and PAM authentication support. See detailed objectives below.
Topic 212	LPI exam 202 prep (topic 212): System security	(This tutorial) Learn how to configure a router, secure FTP servers, configure SSH, and perform various other security administration tasks. See detailed objectives below.
Topic 214	LPI exam 202 prep (topic 214): Network troubleshooting	Coming soon

To start preparing for certification level 1, see the developerWorks tutorials for LPI exam 101. To prepare for the other exam in certification level 2, see the developerWorks tutorials for LPI exam 201. Read more about the entire set of developerWorks LPI tutorials.

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "System security," the sixth of seven tutorials covering intermediate network administration on Linux. In this tutorial, you learn about a wide array of topics related to using Linux as a security-conscious network server. Such issues as routing, firewalls, and NAT translation (and the tools to manage them) are covered, as well as setting security policies for FTP and SSH. You also learn about general access control with tcpd, hosts.allow, and friends (revisiting the discussion in LPI exam 201 prep (topic 209): File and service sharing). Finally, you learn about some basic security monitoring tools, as well as where to find security resources.

As with the other tutorials in the developerWorks 201 and 202 series, this tutorial is intended to serve as a study guide and entry point for exam preparation, rather than complete documentation on the subject. Readers are encouraged to consult LPI's detailed objectives list and to supplement the information provided here with other material as needed.

This tutorial is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weight.

Table 2. System security: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
2.212.2 Configuring a router	Weight 2	Configure a system to perform network address translation (NAT, IP masquerading), and state its significance in protecting a network. This objective includes configuring port redirection, managing filter rules, and averting attacks.
2.212.3 Securing FTP servers	Weight 2	Configure an FTP server for anonymous downloads and uploads. This objective includes precautions to be taken if anonymous uploads are permitted and configuring user access.
2.212.4 Secure shell (SSH)	Weight 2	Configure an SSH daemon. This objective includes managing keys, configuring SSH for users, forwarding an application protocol over SSH, and managing the SSH login.
2.212.5 TCP_wrappers	Weight 1	Configure tcpwrappers to allow connections to specified servers only from certain hosts or subnets.
2.212.6 Security tasks	Weight 3	Install and configure a secure authentication system; perform basic security auditing

	of source code; receive security alerts from various sources; audit servers for open e-mail relays and anonymous FTP servers; install, configure, and run intrusion detection systems; and apply security patches and bug fixes.

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Other resources

As with most Linux tools, it is always useful to examine the manpages for any utilities discussed. Versions and switches might change between utility or kernel version or with different Linux distributions. For more in-depth information, the Linux Documentation Project has a variety of useful documents, especially its HOWTOs. Also, a variety of books on Linux system security have been published; I have found O'Reilly's *TCP/IP Network Administration*, by Craig Hunt to be quite helpful. See the Resources section for links.

Section 2. Configuring a router

About packet filtering

The Linux kernel includes the "netfilter" infrastructure, which enables you to filter network packages. Usually this capability is compiled into the base kernel, but a kernel module may be needed. Either way, module loading should be seamless (for instance, running `iptables` will load `iptables_filter.o` if it needs it).

Packet filtering is controlled with the utility `iptables` in modern Linux systems; older systems used `ipchains`. Before that, it was `ipfwadm`. While you *can* still use `ipchains` in conjunction with recent kernels if backward compatibility is needed, you will almost always prefer the enhanced capabilities and improved syntax in `iptables`. That said, most of the concepts and switches in `iptables` are compatible enhancements to `ipchains`.

Depending on the exact scenario of filtering (firewall, NAT, etc.), filtering and

address translation may occur either before or after routing itself. The same `ipchains` tool is used in either case, but different rule sets ("chains") are used for the cases -- at base, `INPUT` and `OUTPUT`. However, filtering can also affect the routing decision by filtering on the `FORWARD` chain; this may lead to dropping packets rather than routing them.

Routing

As well as filtering with `iptables` (or legacy `ipchains`), the Linux kernel performs *routing* of IP packets it receives. Routing is a simpler process than filtering, though the two are conceptually related.

During routing, a host simply looks at a destination IP address and decides whether it knows how deliver a packet directly to that address or whether a gateway is available that knows how to deliver to that address. If a host can neither deliver a packet itself nor knows what gateway to forward it to, the packet is dropped. However, typical configurations include a "default gateway" that handles every otherwise unspecified address.

Configuration and display of routing information is performed with the utility `route`. However, routing may either be *static* or *dynamic*.

With static routing, delivery is determined by a routing table that is explicitly configured by invocations of the `route` command and its `add` or `del` commands. However, configuring dynamic routing using the `routed` or `gated` daemons that broadcast routing information to adjacent routing daemons is often more useful.

The `routed` daemon supports the Routing Information Protocol (RIP); the `gated` daemon adds support for a number of other protocols -- and can use multiple protocols at once -- such as:

- Routing Information Protocol Next Generation (RIPng)
- Exterior Gateway Protocol (EGP)
- Border Gateway Protocol (BGP) and BGP4+
- Defense Communications Network Local-Network Protocol (HELLO)
- Open Shortest Path First (OSPF)
- Intermediate System to Intermediate System (IS-IS)
- Internet Control Message Protocol (ICMP and ICMPv6)/Router Discovery

Let's look at a fairly typical static routing table:

Listing 1. Typical static routing table

```
% /sbin/route
Kernel IP routing table
```

```
Destination   Gateway         Genmask         Flags Metric Ref  Use Iface
66.98.217.0   *               255.255.255.0   U     0      0      0 eth0
10.10.12.0    *               255.255.254.0   U     0      0      0 eth1
66.98.216.0   *               255.255.254.0   U     0      0      0 eth0
169.254.0.0   *               255.255.0.0     U     0      0      0 eth1
default       ev1s-66-98-216- 0.0.0.0         UG    0      0      0 eth0
```

This means that addresses in the 66.98.217/24 and 66.98.216/23 ranges will be directly delivered over `eth0`. Address ranges 10.10.12/23 and 169.254/16 will be delivered on `eth1`. Anything left over will be sent to the gateway `ev1s-66-98-216-1.ev1servers.net` (the name is cut off in the `route` display; you could also use `route -n` to see that name was IP address 66.98.216.1). If you wanted to add a different gateway for some other address ranges, you might run something like this:

Listing 2. Adding new gateway for other address ranges

```
% route add -net 192.168.2.0 netmask 255.255.255.0 gw 192.168.2.1 dev eth0
```

For a machine that serves as a gateway itself, you will generally want to run dynamic routing, using the `routed` or `gated` daemons, which may supplement a smaller number of static routes. The `routed` daemon is configured by the contents of /etc/gateways. The `gated` daemon is more modern and has more capabilities (as indicated); it is configured by /etc/gated.conf. Generally, if you use either of these, you will want to launch them in your startup scripts. You must *not* run both `routed` and `gated` on the same machine because results will be unpredictable and almost certainly undesirable.

Filtering with iptables

The Linux kernel stores a table of filter rules for IP packets that form a sort of state-machine. Sets of rules that are processed in sequence are known as "(firewall) chains." When one chain meets a condition, one of the possible actions is to shift control to processing another chain as in a state-machine. Before you have added any rules or states, three chains are automatically present: `INPUT`, `OUTPUT`, and `FORWARD`. The `INPUT` chain is where a packet addressed to the host machine passes and potentially from there to a local application process. The `FORWARD` chain is where a packet addressed to a different machine passes, assuming forwarding is enabled and the routing system knows how to forward that packet. A packet generated on the local host is sent into the `OUTPUT` chain for filtering -- if it passes the filters in the `OUTPUT` chain (or any linked chains), it is routed out over its network interface.

One action that a rule can take is to `DROP` a packet; in that case, no further rule processing or state transition is taken for that packet. But if a packet is not dropped, the next rule in a chain is examined to see if it matches the packet. In some cases, satisfaction of a rule will branch process to a different chain and its set of rules. Creation, deletion, or modification of rules and of chains in which rules live is performed with the tool `iptables`. In older Linux systems, the same function was

done using `ipchains` instead. The concepts behind both tools and even for the ancient `ipfwadm` are similar, but `iptables` syntax is discussed here.

A rule specifies a set of conditions that a packet might meet and what action to take if the packet does meet that condition. As mentioned, one common action is to `DROP` packets. For example, suppose you wanted (for some reason) to disable `ping` on the loopback interface (the ICMP interface). You could make this happen with:

Listing 3. Disabling on the loopback interface

```
% iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

Of course, that is a silly rule and we probably want to remove it after we test it, like this:

Listing 4. DROPping the silly rule

```
% iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
```

Deleting a rule with the `-D` option requires either exactly the same options as specified when it was added or specification by rule number (which you *must* determine in the first place) like this:

Listing 5. Specifying rule number so deletion can work

```
% iptables -D INPUT 1
```

A more interesting rule might look at source and destination addresses in packets. For example, suppose that a problem remote network is trying to utilize services on a particular subnet of your network. You might block this on your gateway/firewall machine with:

Listing 6. Blocking the gateway/firewall machine

```
% iptables -A INPUT -s 66.98.216/24 -d 64.41.64/24 -j DROP
```

Doing this will stop anything from the `66.98.216.*` IP block from communicating with anything in the local `64.41.64.*` subnet. Of course, singling out a specific IP block for blacklisting is fairly limited as protection. A more likely scenario might be to *allow* only a specific IP block to access a local subnet:

Listing 7. Letting a specific IP block access a local subnet

```
% iptables -A INPUT -s ! 66.98.216/24 -d 64.41.64/24 -j DROP
```

In this case, *only* the `66.98.216.*` IP block can access the specified subnet.

Moreover, you can use a symbolic name for an address and can specify a particular protocol to be filtered. You can also select a specific network interface (like `eth0`) to filter, but that is less commonly useful. For example, to let only a specific remote network access a local Web server, you might use:

Listing 8. Letting a specific remote network address access a local Web server

```
% iptables -A INPUT -s ! example.com -d 64.41.64.124 -p TCP -sport 80 -j DROP
```

You can specify a number of other options with `iptables`, including rate limits on the number of packets that will be allowed or filtering on TCP flags, for example. See the manpage for `iptables` for more details.

User-defined chains

You have seen the basics of adding rules to the automatic chains. But much of the configurability in `iptables` comes with adding user-defined chains and branching to them if patterns are matched. New chains are defined with the `-N` option; you have already seen branching using the special target `DROP`. `ACCEPT` is also a special target with the obvious meaning. Also, special targets `RETURN` and `QUEUE` are available. The first means to stop processing a given chain and return to its parent/caller. The `QUEUE` handler lets you pass packets to a user-space process for further processing (which might be logging, modification of the packet, or more elaborate filtering than `iptables` supports). The simple example in Rusty Russell's "Linux 2.4 Packet Filtering HOWTO" is a good example of adding a user-defined chain:

Listing 9. Adding a user-defined chain

```
# Create chain to block new connections, except established locally
% iptables -N block
% iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
% iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
% iptables -A block -j DROP  # DROP everything else not ACCEPT'd
# Jump to that chain from INPUT and FORWARD chains
% iptables -A INPUT -j block
% iptables -A FORWARD -j block
```

Notice that the `block` chain `ACCEPT`s in a limited class of cases, then the final rule `DROP`s everything not previously `ACCEPT`ed.

Once you have established some chains, whether adding rules to the automatic chains or adding user-defined chains, you may use the `-L` option to view the current rules.

Network address translation vs. firewalls

The examples we have looked at are mostly in the class of firewall rules. But network address translation (NAT) is also configured by `iptables`.

Basically, NAT is a way of using connection tracking to masquerade packets coming from a local subnet address as the external WAN address before sending them out "over the wire" (on the `OUTPUT` chain). The gateway/router that performs NAT needs to remember which local host connected to which remote host and reverse the address translation if packets arrive back from the remote host.

From a filtering perspective though, you simply pretend that NAT does not exist. The rules you specify should simply use the "real" local addresses regardless of how NAT might masquerade them to the outside world. Enabling masquerading, such as in basic NAT, just uses the below `iptables` command. To use this you will need to make sure the kernel module iptables_nat is loaded and also turn on IP forwarding:

Listing 10. Enabling the masquerade

```
% modprobe iptables_nat    # Load the kernel module
% iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
% echo 1 > /proc/sys/net/piv4/ip_forward   # Turn on IP forwarding
```

This capability is called *source NAT* -- the address of the outgoing packet is modified. *destination NAT* (DNAT) also exists to enable port forwarding, load sharing, and transparent proxying. In those cases, incoming packets are modified to get to the relevant local host or subnet.

But most of the time when users or administrators talk about NAT, they mean source NAT. If you mean to configure destination NAT, you would specify `PREROUTING` rather than `POSTROUTING`. For DNAT, the packets are transformed before they are routed.

Section 3. Securing FTP servers

FTP servers

Many different FTP servers are available for Linux, and different distributions offer different servers. Naturally, configuration of different servers varies, though most tend to follow similar configuration directives.

A popular FTP server is vsftpd (the Very Secure FTP daemon). ProFTP is also in wide use, as are wu-ftpd and ncftpd.

For many purposes, FTP is not really needed at all. For example, secure transfers for users who have accounts on a server machine can often be accomplished using `scp` (secure copy), which relies on the underlying SSH installation, but otherwise mostly mimics the familiar `cp` command.

The configuration file for vsftpd is /etc/vsftpd.conf. Other FTP servers use similar

files.

FTP configuration options

Here are a few options to keep in mind in /etc/vsftpd.conf (and probably in your server if you use a different one):

- `anonymous_enabled`: Lets anonymous users log in using the usernames "anonymous" or "ftp".
- `anon_mkdir_write_enable`: Lets anonymous users create directories (within world-writable parent directories).
- `anon_upload_enable`: Lets anonymous users upload files.
- `anon_world_readable_only`: "YES" by default; rarely a good idea to change it. Only lets anonymous FTP access world-readable files.
- `chroot_list_enable`: Specifies a set of users (listed in /etc/vsftpd.chroot-list) in a "chroot jail" in their home directory upon login.
- `ssl_enable`: Supports SSL-encrypted connections.

Read the manpages for your FTP server for more complete options. Generally, running an FTP server is as simple as tweaking a configuration file and running the server within your initialization scripts.

Section 4. Secure shell (SSH)

Client and server

Most every Linux machine (as well as most other operating systems) should have a secure shell (SSH) client. Often, the OpenSSH version is used, but a variety of compatible SSH clients are sometimes used. While an SSH client is essential to connect to a host, the larger security issues arise in properly configuring an SSH server.

Since a client initiates a connection to a server, the client is actively choosing to trust the server. Just having an SSH client does not allow any kind of access *into* a machine; therefore, it does not expose vulnerabilities.

Configuring a server is also not particularly complex; the server daemon is designed to enable and enforce good security practices. But clearly it is a server that is sharing resources with clients based on requests from the clients the server decides to honor.

The SSH protocol has two versions, version 1 and version 2. In modern systems, using protocol version 2 is always preferred, but generally both clients and servers maintain backward compatibility with version 1 (unless this capability is disabled with configuration options). This lets you connect to increasingly uncommon version 1-only systems.

Somewhat different configuration files are used in version 1 and version 2 protocols. For protocol version 1, a client first creates an RSA key pair using ssh-keygen and stores the private key in $HOME/.ssh/identity and the public key in $HOME/.ssh/identity.pub. This same identity.pub should be appended to the remote $HOME/.ssh/authorized_keys files.

Obviously, there is a chicken-and-egg problem here: How can you copy a file to a remote system before you have access? Fortunately, SSH also supports a fallback authentication method of sending encrypted-on-the-wire passwords that are evaluated through the usual remote-system login tests (such as, the user account must exist, and the right password must be provided).

Protocol 2 supports both RSA and DSA keys, but RSA authentication is somewhat enhanced rather than identical to that in protocol 1. For protocol 2, private keys are stored in $HOME/.ssh/id_rsa and $HOME/.ssh/id_dsa. Protocol 2 also supports a number of extra confidentiality and integrity algorithms: AES, 3DES, Blowfish, CAST128, HMAC-MD5, HMAC-SHA1, and so on. The server can be configured as to preferred algorithms and order of fallbacks.

For general configuration options rather than key information, the client stores its keys in /etc/ssh/ssh_config (or if available, in /$HOME/.ssh/config). Client options can also be configured with the `-o` switch; a particularly common switch is the `-X` or `-x` to enable or disable X11 forwarding. If enabled, the X11 port is tunneled through SSH to enable encrypted X11 connections.

Tools like `scp` also use similar port forwarding over SSH. For example, on the local machine I am working on, I can launch onto the local display an X11 application that only exists remotely (on my local subnet in this case):

Listing 11. Launching a remote X11 application

```
$ which gedit  # not on local system
$ ssh -X dqm@192.168.2.2
Password:
Linux averatec 2.6.10-5-386 #1 Mon Oct 10 11:15:41 UTC 2005 i686 GNU/Linux
No mail.
Last login: Thu Feb 23 03:51:15 2006 from 192.168.2.101
dqm@averatec:~$ gedit &
```

Configuring the server

The `sshd` daemon, specifically the OpenSSH version, enables secure encrypted communications between two untrusted hosts over an unsecured network. The base `sshd` server is normally started during initialization and listens for client connections

forking a new daemon for each client connection. The forked daemons handle key exchange, encryption, authentication, command execution, and data exchange.

As with the client tool, the `sshd` server accepts a variety of options on the command line, but is normally configured by the file /etc/ssh/sshd_config. A number of other configuration files are also used. For example, the access controls /etc/hosts.allow and /etc/hosts.deny are honored. Keys are stored in a similar fashion to the client side, in /etc/ssh/ssh_host_key (protocol 1), /etc/ssh/ssh_host_dsa_key, /etc/ssh/ssh_host_rsa_key, and public keys in /etc/ssh/ssh_host_dsa_key.pub and friends. Also, as with the client, you will use ssh-keygen to generate keys in the first place. See the manpage for `sshd` and `ssh-keygen` for details on configuration files and copying generated keys to appropriate files.

Many configuration options ar in /etc/ssh/sshd_config, and the default values are generally sensible (and sensibly secure). A few options are worth mentioning:

- `AllowTcpForwarding` enables or disables port forwarding (tunneling), and is set to "YES" by default.
- `Ciphers` controls the list and order of encryption algorithms to be utilized.
- `AllowUsers` and `AllowGroups` accept wildcard patterns and allow you to control which users may even attempt further authentication.
- `DenyGroups` and `DenyUsers` act symmetrically as you would expect.
- `PermitRootLogin` lets the root user SSH into a machine.
- `Protocol` lets you specify whether both protocol versions are accepted (and if not, which one is).
- `TCPKeepAlive` is good to look at if you are losing SSH connections. A "keepalive" message is sent to check connections if this is enabled, but this can cause disconnection if transient errors occur in the route.

SSH tunneling

OpenSSH lets you create a tunnel to encapsulate another protocol within an encrypted SSH channel. This capability is enabled on the `sshd` server by default, but could have been disabled with command-line or configuration-file options. Assuming the capability is enabled, clients can easily emulate whatever port/protocol they wish to use for a connection. For example, to create a tunnel for telnet:

Listing 12. Digging a tunnel for telnet

```
% ssh -2 -N -f -L 5023:localhost:23 user@foo.example.com
% telnet localhost 5023
```

Of course, this example is fairly pointless since an SSH command shell does the same thing as a telnet shell. But you could create a POP3, HTTP, SMTP, FTP, X11,

or other protocol connection in this analogous manner. The basic concept is that a particular local host port acts as if it were the remote service with actual communication packets traveling over the SSH connection in encrypted form.

The options we used in the example are:

- `-2` (use protocol 2),
- `-N` (no command/tunnel only),
- `-f` (SSH in background), and
- `-L,` (describe tunnel as "localport:remotehost:remoteport").

The server (with username) is also specified.

Section 5. TCP_wrappers

What is "TCP_wrappers"?

The first thing to know about TCP_wrappers is that you *should not* use it, and it is not actively maintained. However, you might find the tcpd daemon from TCP_wrappers still running on a legacy system. In its time, this was a good application, but its functionality has been superceded by iptables and other tools. The general purpose of TCP_wrappers is to monitor and filter incoming requests for SYSTAT, FINGER, FTP, TELNET, RLOGIN, RSH, EXEC, TFTP, TALK, and other network services.

TCP_wrappers can be configured in a couple of ways. One is to substitute tcpd for other servers, providing arguments to pass control on to the particular server once tcpd has done its logging and filtering. Another method leaves the network daemons alone and modifies the inetd configuration file. For example, an entry such as:

```
tftp dgram udp wait root /usr/etc/tcpd in.tftpd -s /tftpboot
```

causes an incoming tftp request to run through the wrapper program (tcpd) with a process name `in.tftpd`.

Section 6. Security tasks

This objective is a hodgepodge of tasks -- all important for maintaining a secure

network -- that I can't hope to do justice to in the space of this tutorial. I recommend spending time familiarizing yourself with the resources and tools listed in this section.

Web sites worth monitoring for security issues and patches include:

- Security focus news: The Security Focus Web site is one of the best sites for reporting and discussing security issues and specific vulnerabilities. The site includes several newsletters and alerts that you can subscribe to, as well as general columns and searchable bug reports.
- The Bugtraq mailing list: A full-disclosure moderated mailing list for the *detailed* discussion and announcement of computer security vulnerabilities: what they are, how to exploit them, and how to fix them.
- CERT Coordination Center: Hosted by Carnegie Mellon University, CERT has a range of advisories similar to the Security Focus site, with a bit more emphasis on tutorials and guidelines. Keeping track of multiple such sites is a good way to make sure you are current on all the security incidents affecting your OS, distribution, and specific tools or servers.
- Computer Incident Advisory Capability: CIAC Information Bulletins are distributed to the Department of Energy community to notify sites of computer security vulnerabilities and recommended actions. Similarly, CIAC Advisory Notices serve to alert sites to severe, time-critical vulnerabilities and solutions to be applied as soon as possible. CIAC Technical Bulletins cover technical security issues and analyses of a less time-sensitive nature.
- Information on securing open mail relays: A common vulnerability on systems with mail servers is failure to properly secure systems against malicious use by spammers and fraudulent mailers. The Open Relay Database provides tutorials on security-particular mail tools, open relay testing online tools, and a database of known problem servers that can be used to configure blacklists if site administrators so desire.

Tools to monitor security you might consider running include:

- Open Source Tripwire: A security and data integrity tool for monitoring and alerting on specific file changes.
- `scanlogd` : A TCP port scan detection tool.
- Snort: Network intrusion prevention and detection using a rule-driven language; it uses signature-, protocol-, and anomaly-based inspection methods.

Resources

Learn

- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- *TCP/IP Network Administration, Third Edition* by Craig Hunt (O'Reilly, April 2002) is an excellent resource on Linux networking.
- For more in-depth information, the Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- In the developerWorks Linux zone, find more resources for Linux developers.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- This list of more than 700 Linux User Groups around the world can help you find local and distance study groups for LPI exams.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

David Mertz
David Mertz has been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* . For more on David, see his personal Web page.

Trademarks

DB2, Lotus, Rational, Tivoli, and WebSphere are trademarks of IBM Corporation in the United States, other countries, or both.
Linux is a trademark of Linus Torvalds in the United States, other countries, or both.